IMGD Master's Project Report

PolyCreate– Enabling Live Coding in Unreal Game Engine

By: Utsav Sanghvi Fall 2023

Project Advisor: Professor Charlie Roberts

Reader: Professor Rose Bohrer

Table of Contents

1. Introduction

PolyCreate aims to broaden live coding practices by integrating a dynamic programming environment with the powerful Unreal Game Engine. This integration provides developers with a versatile platform to experiment, create, and visualize their code in real-time. Developers can manipulate shapes, apply transformations, and experiment with visual effects, all within an immersive real-time environment. At the heart of PolyCreate lies the Unreal Game Engine, known for its power and versatility. At the same time, live coding has become an increasingly popular method of performance, where audiovisual works are programmed on stage and code is projected for audience members to follow [13]. The PolyCreate Project seeks to combine the strengths of Unreal Engine with the fluidity of live coding.

PolyCreate is centered around *LiveBuild*, a domain-specific programming language controlling the Unreal Engine and made for live coding. LiveBuild lets users easily tweak and play with graphics in real-time, creating a dynamic and engaging environment for creativity. LiveBuild was created using the Parsing Expression Grammar language formalism [14], using an editor that runs in the browser.

Image 1. LiveBuild editor environment

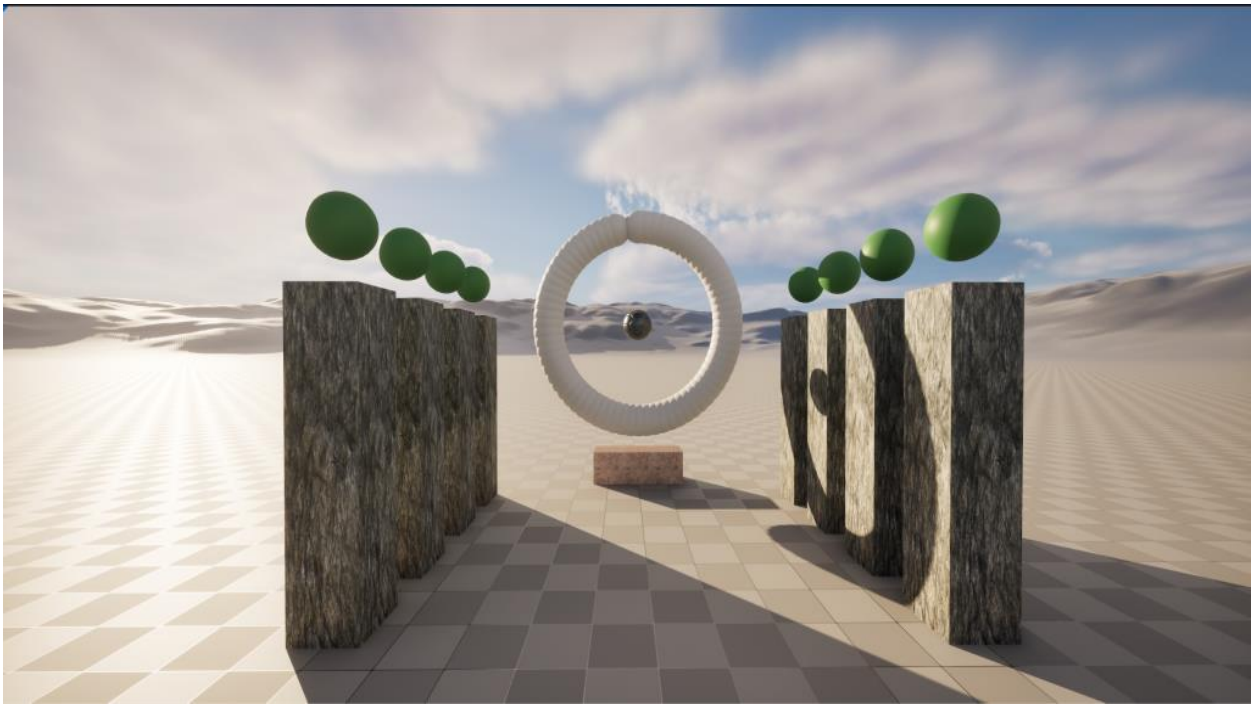

Image 2. "Altar of Creation" made using the code shown in Image 1, created during user testing of the system.

In this paper, we begin by discussing four live coding environments for graphics and games programming. We then describe how we created the PolyCreate system, and conclude with a discussion of our user testing and future work for the PolyCreate project.

2. Literature Review

2.1 Exploring Live Coding Environments in Game Development

In this section, we describe a few of the diverse tools available to developers who are interested in live coding graphics. From Hydra's mesmerizing visual effects to Arcadia's fusion of Clojure and Unity, each platform offers a unique approach to live coding within the context of game development. However, alongside their strengths, these platforms also present a range of challenges that developers must navigate.

2.2 Insights from Existing Live Coding Environments

Our investigation delved into several live coding environments, including Hydra, Arcadia for Unity, the wgsl_live environment, and ENU for Godot Engine. Each of these platforms offers distinct features and capabilities.

Arcadia for Unity

Arcadia bridges the gap between the Clojure programming language and the Unity 3D game engine, enabling developers to leverage a functional programming paradigm within Unity. This integration opens up new avenues for creativity and expressiveness in game development. However, Arcadia's main challenge lies in its steep learning curve. Developers accustomed to traditional game development workflows may find it challenging to adapt to the functional programming paradigm, limiting Arcadia's accessibility to a niche audience [8].

Hydra

Hydra is well-known for its ability to create stunning visual effects in real-time. It's a place where live coders can easily manipulate graphics using math and algorithms to make mesmerizing visuals. We liked how Hydra keeps things simple, so we wanted to bring that same simplicity to Unreal Engine. Our aim was to make a language that's easy for anyone to use, inspired by Hydra's user-friendly style.[7]

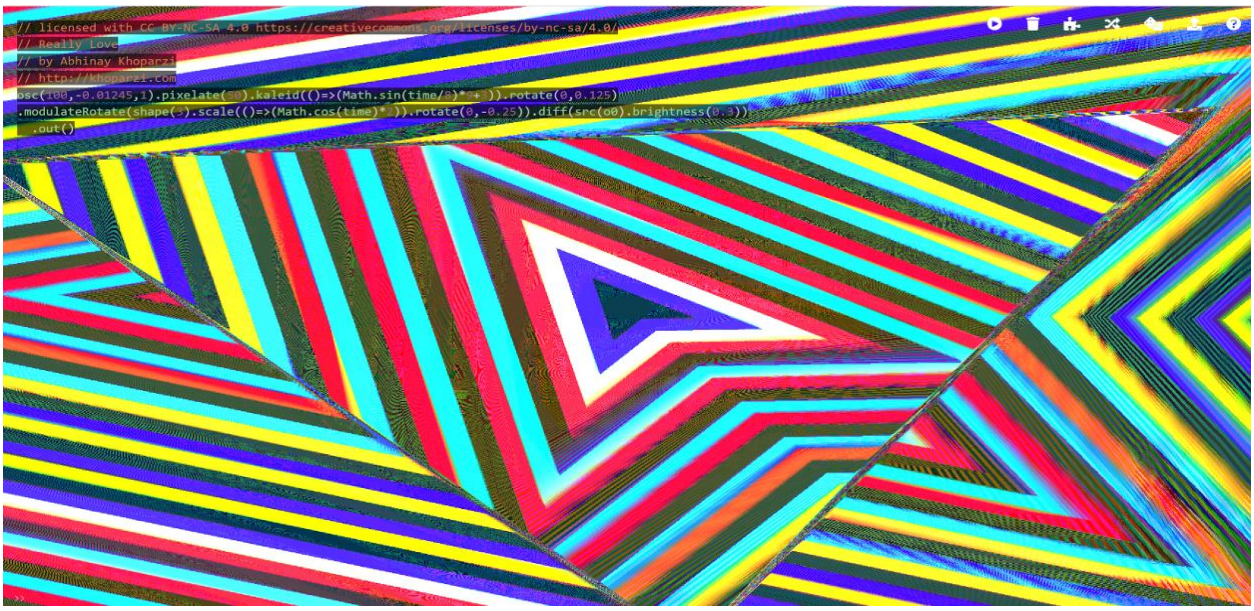

Image 3. Hydra Live coding environment

WGSL Live Environment

The WGSL Live Environment stands out for its technical prowess in graphics programming. It provides developers with advanced tools and features to manipulate graphics and create complex visual effects in real-time. Despite its technical capabilities, WGSL Live Environment faces usability challenges that hinder its widespread adoption. Developers may struggle to understand the intricacies of the low-level graphics programming used by this system, potentially leading to a steep learning curve [9].

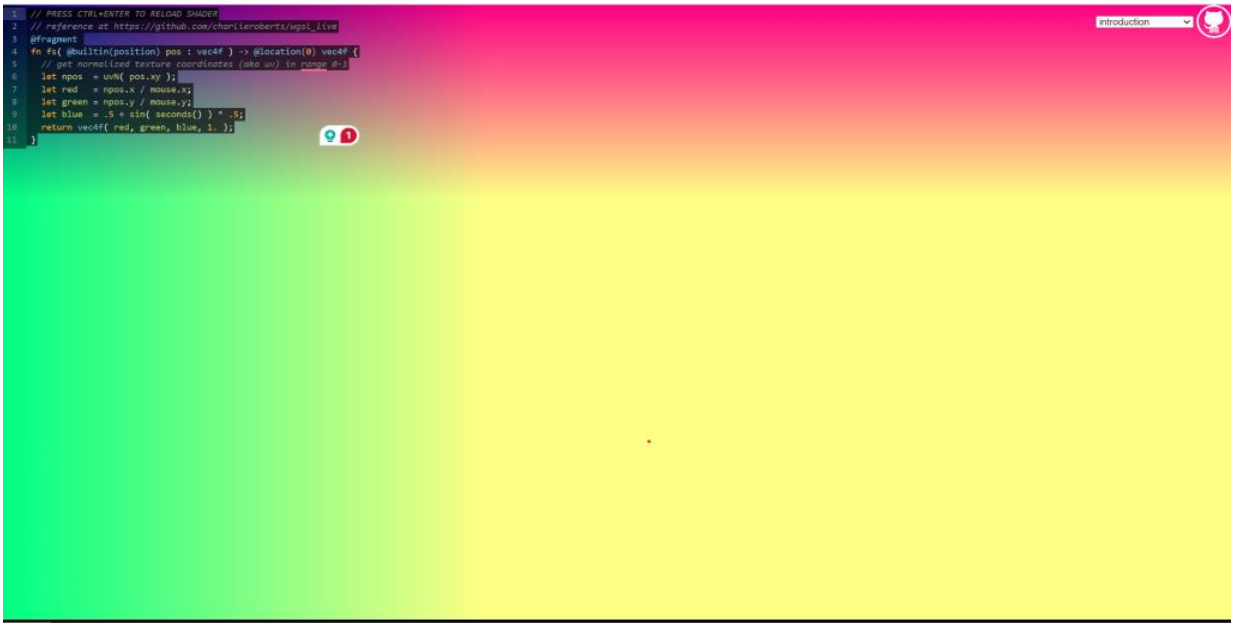

Image 4. WGSL Live coding environment

ENU for Godot Engine

ENU is tailored specifically for the Godot Engine, offering developers the ability to engage in real-time iteration and experimentation. It provides a platform for developers to tweak and modify game elements on-the-fly, facilitating rapid prototyping and iteration. However, we believe ENU faces usability challenges that impact its accessibility. Developers may encounter difficulties in navigating the interface and understanding the nuances of live coding within the Godot Engine environment [10].

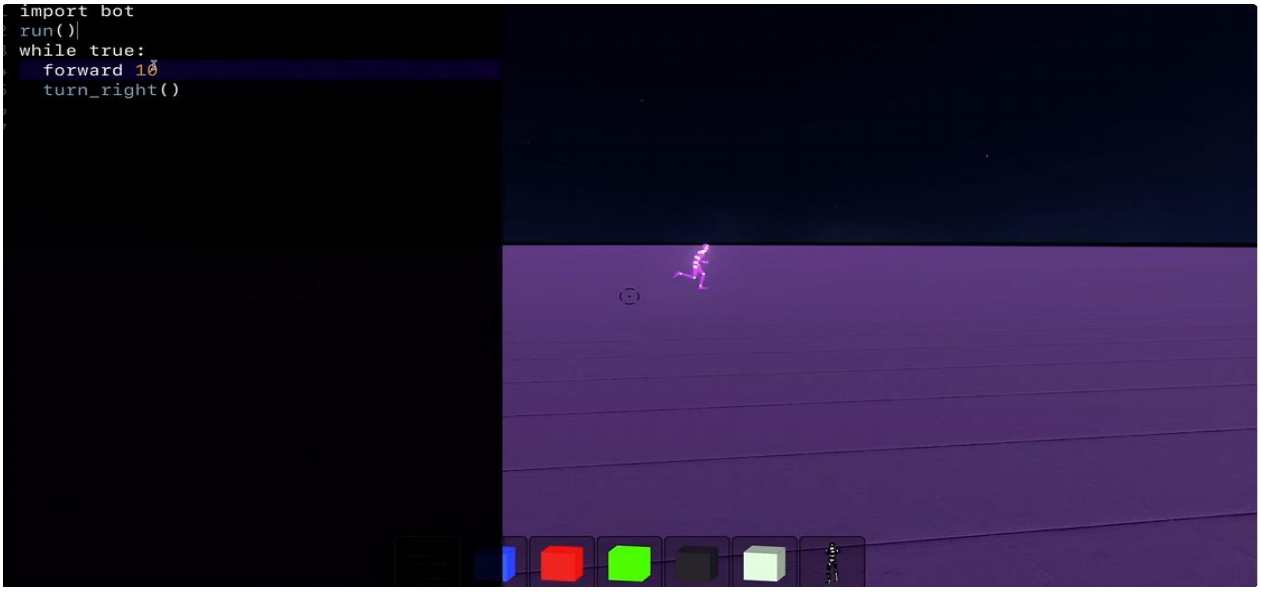

Image 5. Enu Live coding environment

3. Procedure:

The heart of the PolyCreate project lies in a browser-based live-coding environment that integrates with Unreal Engine.

3.1 Open Sound Control Message

The integration of Open Sound Control (OSC) [15] enables communication between the browser-based live coding environment and the Unreal Engine. However, due to security limitations of networking in the browser, it cannot send OSC messages directly; instead we send WebSocket messages to a custom node.js server, which then converts these messages to the Open Sound Control protocol. These messages are received by the Unreal Engine, which then take appropriate actions based on message content. Despite requiring an additional server, we feel this approach is still easier than trying to integrate support for WebSocket messaging into Unreal directly.

The official documentation provided by Epic outlines the process of integrating OSC into the Unreal Engine environment [4] [5]. This involves importing the OSC plugin directly into our project and configuring an Actor within the engine to act as the OSC Receiver. This specialized Actor is tasked with listening for OSC messages sent to a specific port, effectively serving as the gateway through which communication between the browser and the engine takes place.

Once an OSC message is received by the designated OSC Receiver, the Unreal Engine springs into action, executing a predefined set of instructions to parse the message and determine the appropriate response. This could entail a variety of actions, such as printing information to the console for debugging purposes or triggering specific in-game events based on the content of the message.

By leveraging OSC, we're able to create a dynamic and interactive environment where developers can control and manipulate the Unreal Engine in real-time using our custom language LiveCreate, as described in Section 3.4.

3.2 Unreal Engine

We built a library of digital assets within the engine. This library acts as a creative toolkit, housing a range of meshes, animations, particle effects, and more. When an OSC message arrives at the Unreal Engine's doorstep, it sets off a chain reaction of events managed by the OSC Receivers within the engine. These receivers take incoming messages and decipher their contents to determine the best course of action. At the core of this operation are the OSC Managers, specialized Actors that schedule the execution

of commands within the game environment. They interpret OSC messages and translate them into tangible actions within the scene.

Each asset within the Unreal Engine is encapsulated within an Actor blueprint, complete with its own unique identification (UID) and parameters. By specifying the UID value in OSC messages, developers can control every aspect of the virtual world, empowering them to fine-tune and manipulate their creations.

Once the specified objects are identified within the scene, the Unreal Engine springs into action, executing a series of predefined operations based on the information gleaned from the OSC messages. This dynamic process encompasses a myriad of tasks, including precise placement, fluid movement, graceful rotation, immersive animation playback, intricate material adjustments, seamless deletion, and much more. It's this seamless integration of OSC communication and asset manipulation that empowers developers to unleash their creativity and craft truly immersive and interactive experiences within the Unreal Engine environment.

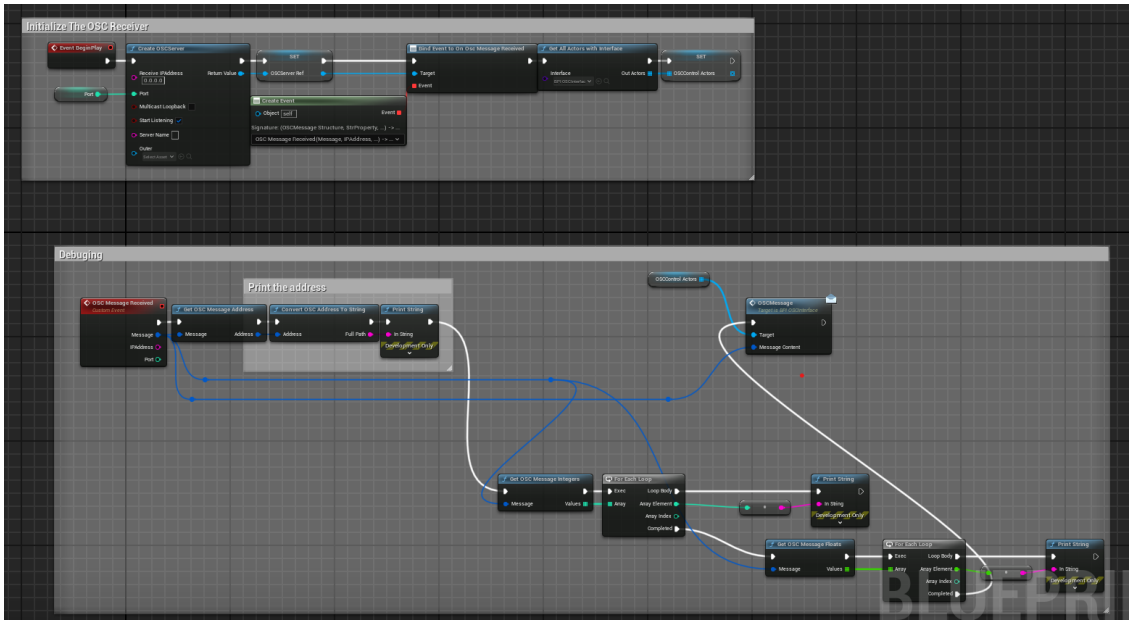

Image 6. OSC Receiver, part 1

At the core of our script lies the "Event to OSCMessage" node, serving as the primary hub for executing the script's actions. This node plays a crucial role in capturing messages transmitted through the Open Sound Control (OSC) protocol. OSC is a versatile communication protocol commonly seen in multimedia performances, facilitating real-time data exchange with the Unreal Engine. Its adaptability to handle various data types makes it a preferred choice for interactive systems aiming for seamless integration.

Once a message is received, the "Get OSC Message Address" node steps in to extract the address pattern, akin to a URL, from the incoming data. This pattern essentially outlines

the message's structure, indicating its type or purpose. Subsequently, the extracted address undergoes a conversion into a string format for simplified comparison and manipulation, courtesy of the "Convert OSC Address To String" node.

A pivotal component in our setup is the "Switch on String" node, acting as a control center to guide the script's execution based on the value of the address string. Each output from this node corresponds to a specific command, identified by string literals like /create/cube or /transform/object. This dynamic routing mechanism underscores the versatility of OSC, seamlessly accommodating various interaction models.

Attached to each output of the "Switch on String" node are function nodes, housing detailed instructions pertinent to Unreal Engine's game objects. These functions, such as Create Object or Transform Object, execute tasks aligned with the associated command, be it spawning new entities or adjusting their spatial properties within the virtual realm.

An intriguing aspect is the inclusion of a "SET" node within each function node, assigning a unique "Command Tag." These tags offer valuable metadata linked to each command, potentially aiding in logging or debugging processes. This added layer of context enhances the clarity and efficiency of command execution, contributing to a smoother workflow.

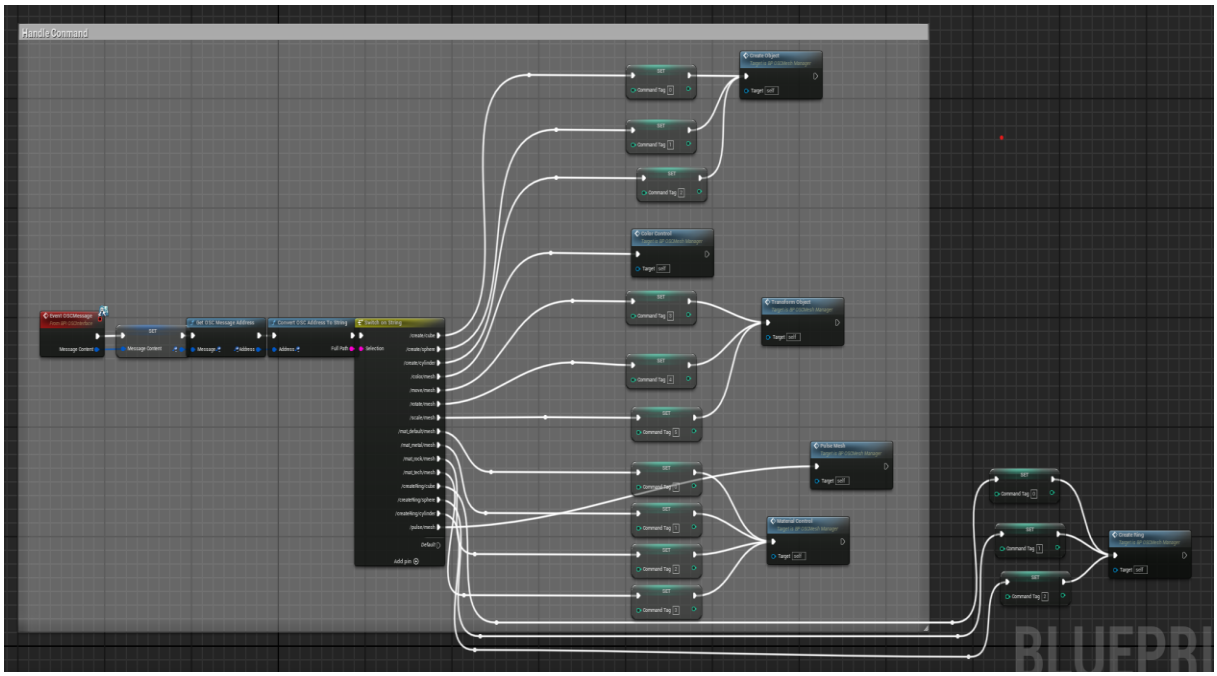

Image 7. Command Handler

Central to our event handling mechanism is the "Switch on String" node, responsible for directing the script's execution based on the syntax of incoming messages. When the

syntax matches one of the cases defined within the "Switch on String" node, the corresponding event is triggered.

Subsequently, several custom events come into play to ensure smooth processing of the incoming data. Firstly, we utilize a "Set Vector" custom event to manage vector data associated with the incoming message. Additionally, a "Get Object ID" custom event is employed to validate whether the user has provided any arguments along with the message.

The "Get Object ID" event plays a crucial role in checking the integrity of the provided arguments. It verifies whether the user has entered an ID and whether the entered value is an integer, as expected. Arguments supplied by the user are assumed to be floating-point values, whereas the ID tag specifically requires an integer value for proper identification within the system.

By implementing these custom events and validation checks, we ensure robust handling of incoming data, enhancing the reliability and accuracy of our event-driven system.

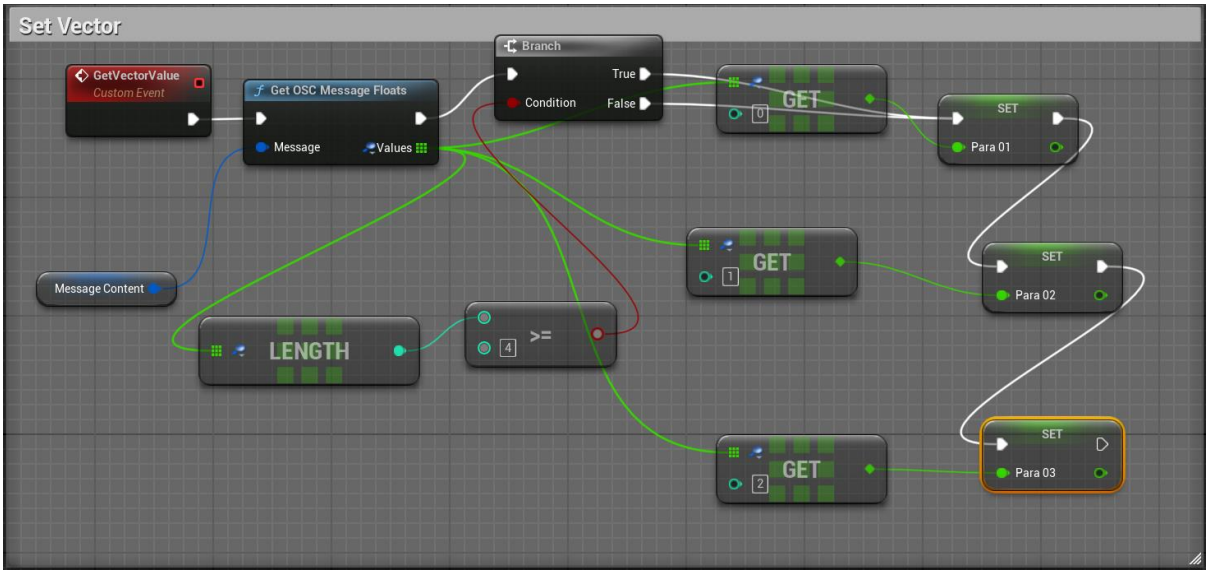

Image 8. Set Vector custom event

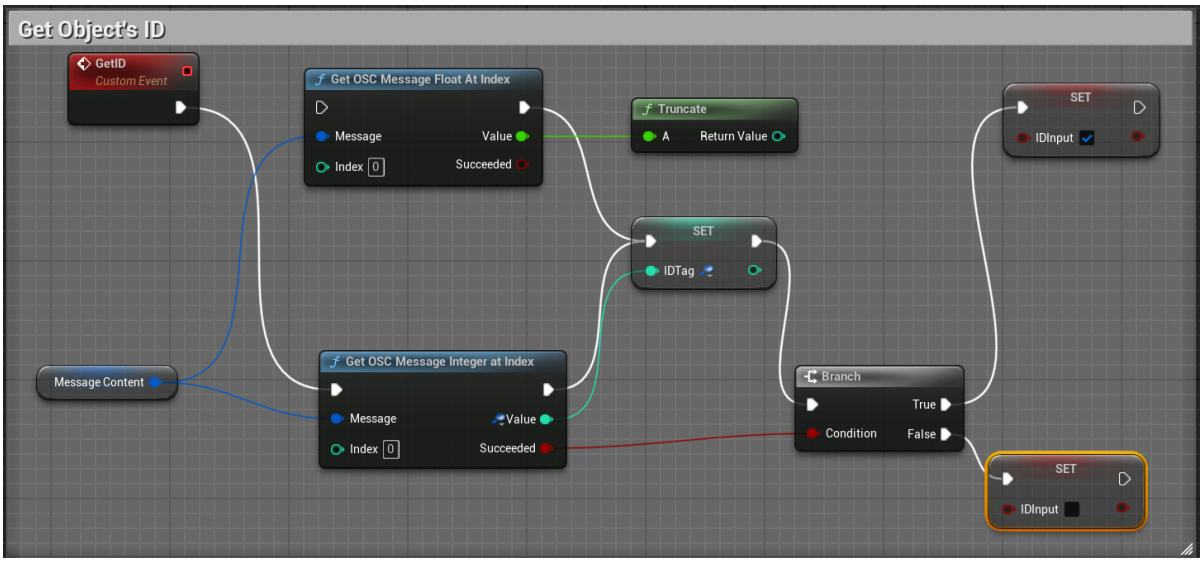

Image 9. Get Object ID custom event

3.3 Features of PolyCreate:-

1. Create: Cube, Sphere, Cylinder

PolyCreate empowers users to bring their ideas to life by offering a range of foundational shapes: cubes, spheres, and cylinders. PolyCreate's intuitive creation tools make it easy to generate and customize these essential geometric primitives.

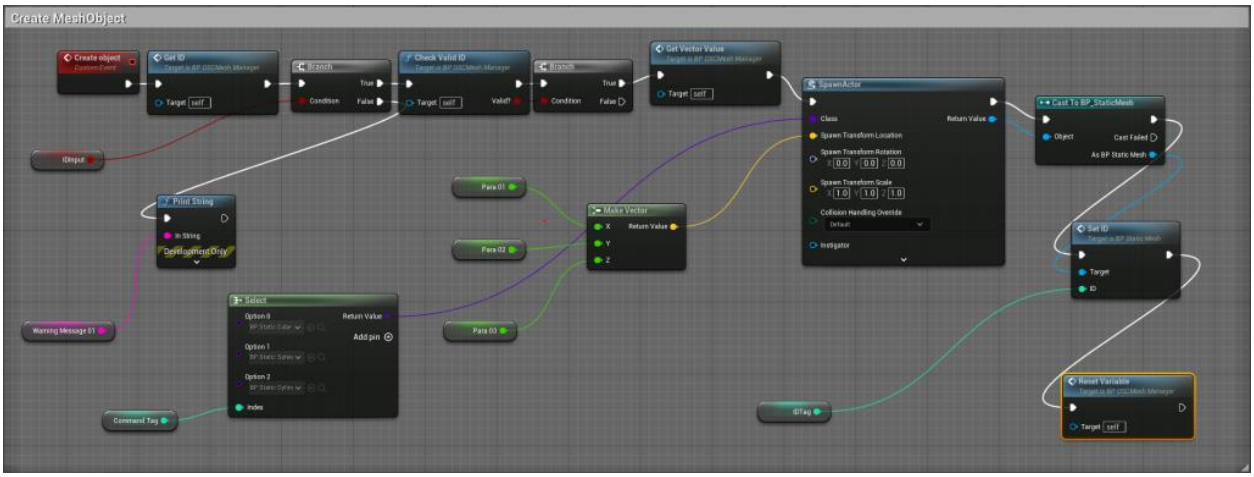

Image 10. Creating mesh

2. Manipulate: Scale, Color, Rotate

PolyCreate enables users to scale, position, and rotate objects to precise dimensions and locations. Users can also manipulate the color of meshes in addition to controlling their material properties.

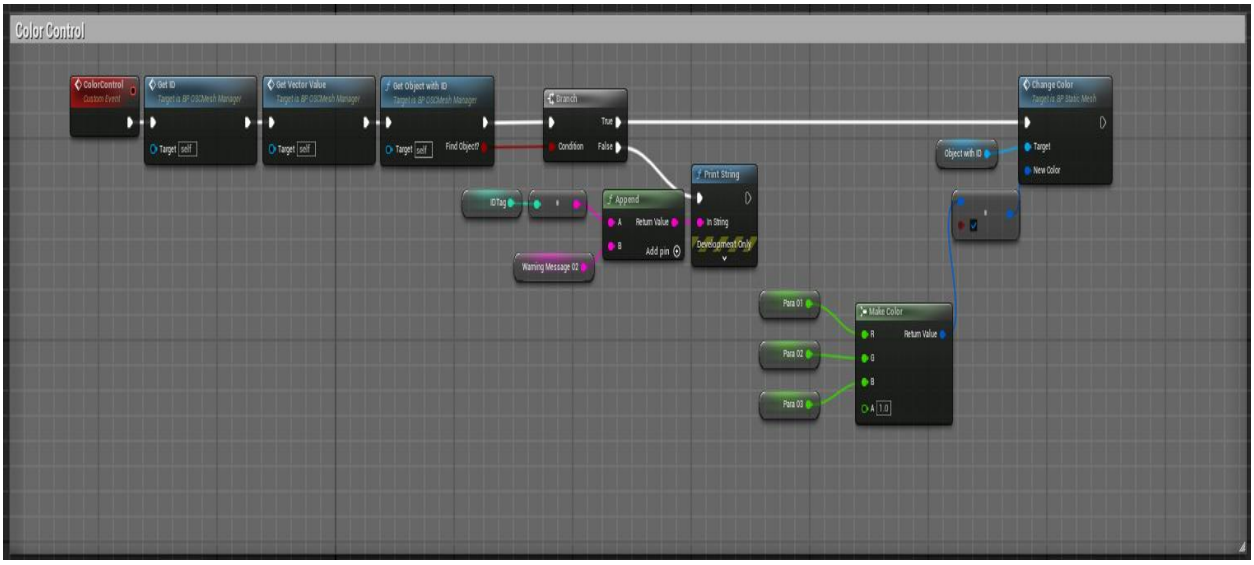

Image 11. Color control mesh

3. Material Modification

PolyCreate's material modification features enables users to fine-tune the appearance of objects by adjusting textures, reflectivity, transparency, and more.

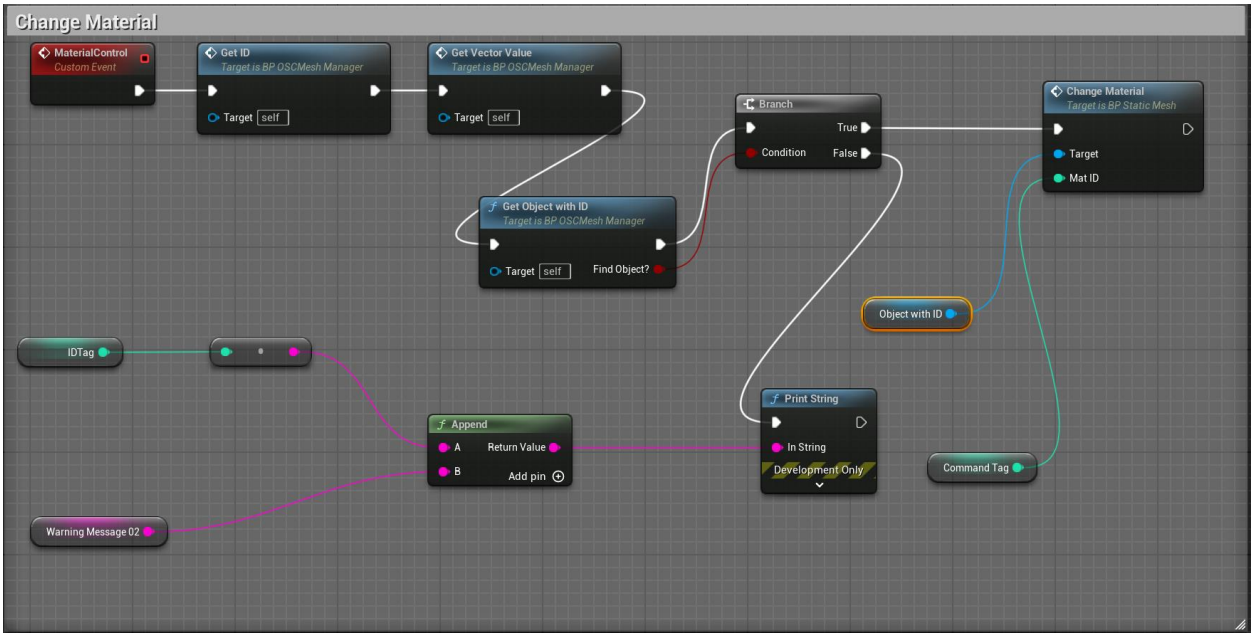

Image 12. Change material of the mesh

4. Fire Effects: Create, Deactivate

Users can easily toggle simple fire effects on and off.

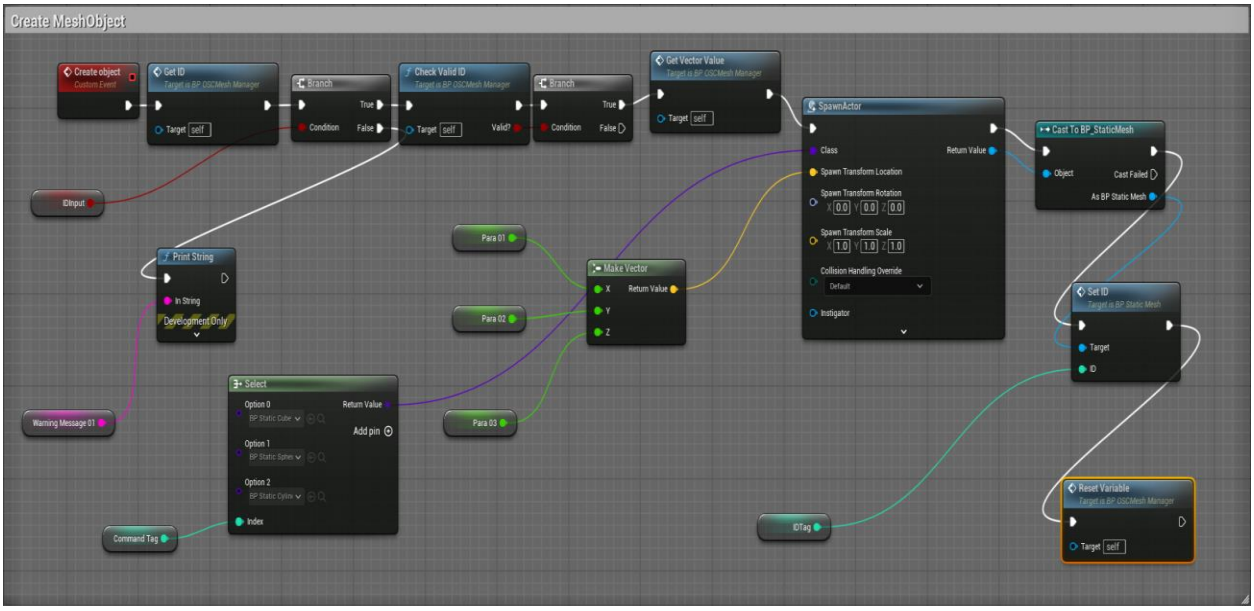

Image 13. Create and deactivate fire

5. Ring Creation: Cube, Sphere, Cylinder

PolyCreate makes it simple to generate rings of objects with precision. Users can choose from cubes, spheres, or cylinders to create formations that add depth and visual interest to scenes.

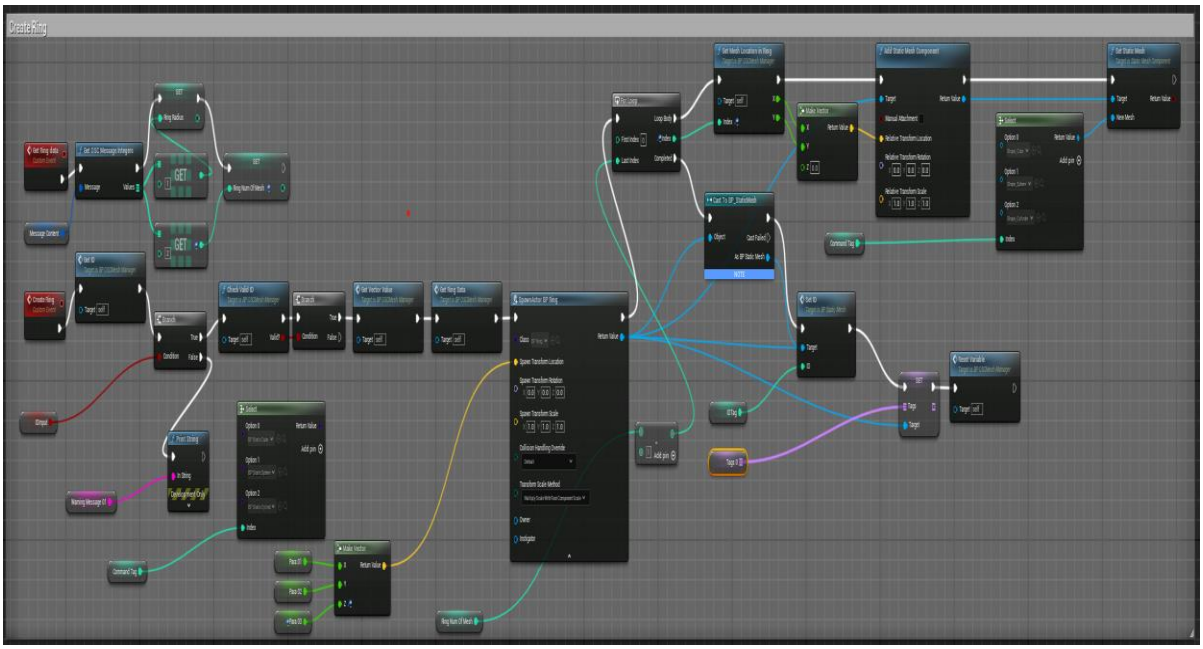

Image 14. Creating ring of particular shape

6. Animation: Pulse Command

PolyCreate's pulse command adds rhythmic pulsations to objects, creating dynamic visual effects that draw the eye.

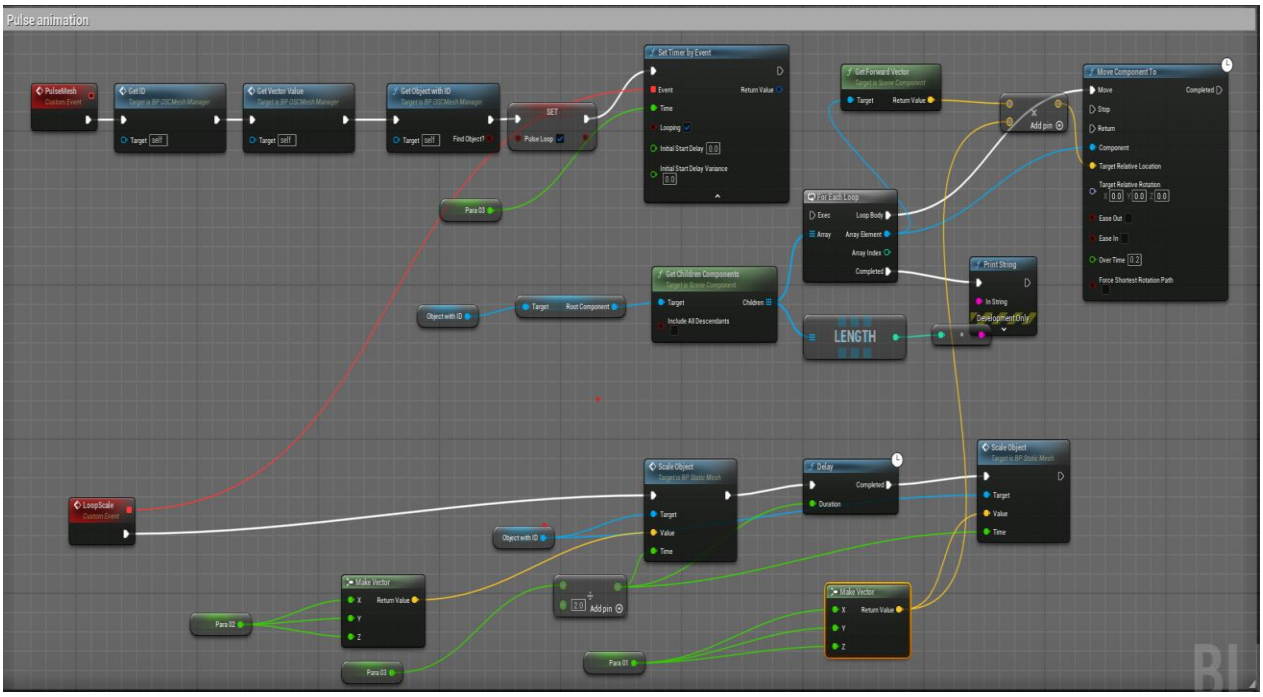

Image 15. Pulse animation

3.4 LiveBuild Language

The LiveBuild language consists of a set of instructions designed to manipulate internal assets within the Unreal Engine. We standardized these instructions to enhance human comprehensibility. Presently, the syntax structure of LiveBuild can be summarized as follows: *Command + Target + ID (optional) + arguments (optional)*. For instance, the command "create cube 1 0.0 0.0 0.0" signifies the creation of a cube in the current level of the Unreal Engine, positioned at the world coordinates (0.0, 0.0, 0.0), with an assigned ID of 1.

The code within LiveBuild, or its instruction set, undergoes transformation into OSC messages via a server. While the LiveBuild language outputs WebSocket messages, we decided to transform these messages into OSC messages to make them easier for Unreal to parse. We created a custom web server in Node.js that takes the WebSocket messages from LiveBuild, converts them to OSC messages, and then forwards them to the Unreal application. Subsequently, the server dispatches the converted OSC messages to a designated port. If the currently running Unreal project has an OSC Message Receiver configured to listen on this port, the engine receives this information and responds accordingly. Through this workflow, we achieve the capability to control a running Unreal project externally via the browser.

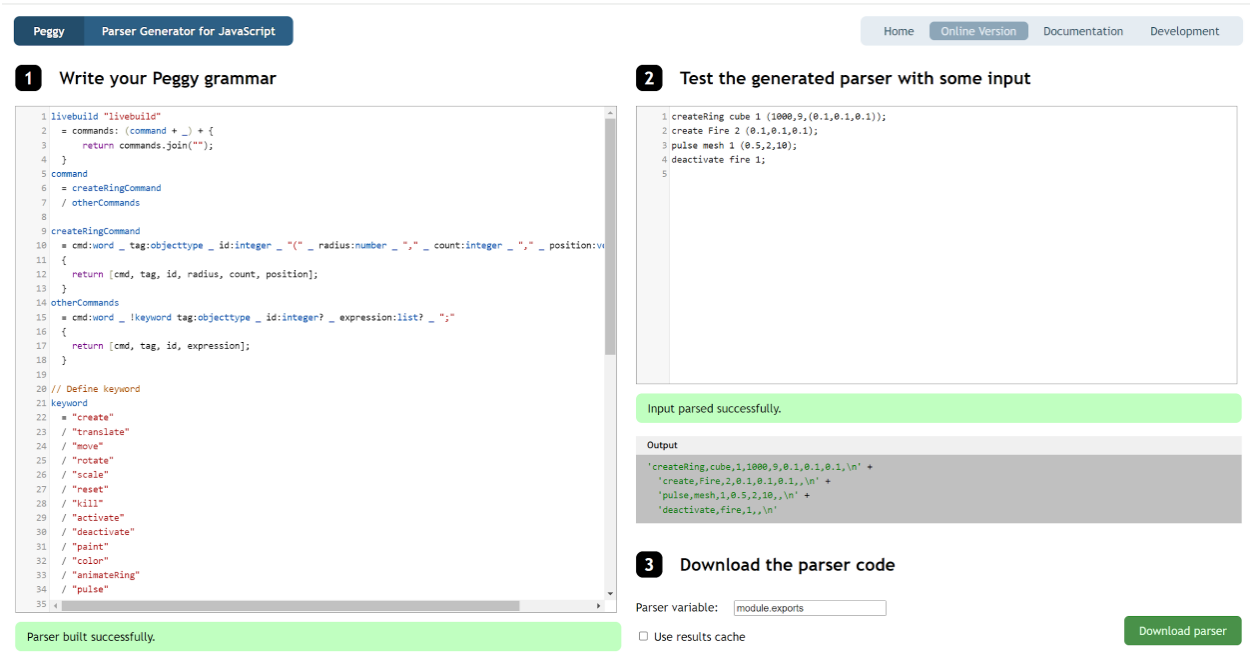

Image 16.peg.js parser file for language

Exception Handling

In this project, errors and exceptions primarily stem from the LiveBuild programming language itself. Users must ensure the validity of their input, meaning that the provided instructions adhere to the prescribed syntax structure. Once the instructions are transformed into OSC messages and received by an OSC Receiver within the Unreal Engine, the engine processes the OSC messages regardless of the logical coherence of the given instructions. If the provided instruction aligns with a predefined scenario, the engine responds accordingly. In cases where the instruction does not match any preset scenarios, the engine displays "No reaction" on the screen.

4. User Evaluation

Our evaluation strategy draws on a range of methodologies to assess the usability and effectiveness of the PolyCreate system. Using techniques outlined in [1] we've conducted an evaluation that encompasses various approaches. Our testing was performed with WPI students no prior experience with LiveBuild. By observing their interactions and collecting feedback, we gained valuable insights into how intuitive and user-friendly LiveBuild is for beginners.

In addition to survey data, direct observation played a crucial role in understanding user behavior. By employing methods like the talk aloud protocol [12], we observed users as they navigated LiveBuild in real-time. This observational approach complemented survey findings, providing us with deeper insights into user preferences and pain points.

4.1 Results and Insights

In this section, we discuss the insights gleaned from our Talk Aloud Protocol sessions, where users engaged in the process of building scenes using PolyCreate.

Task Execution

During the Talk Aloud protocol sessions, users created scenes with PolyCreate, guided by provided instructions. They began by placing cubes, spheres, and cylinders, followed by the manipulation of size, color, and material properties to improve visual aesthetics. Users strategically integrated dynamic elements such as fire effects and animated pulsing rings to infuse life into their compositions.

Several noteworthy insights emerged:

*Explorative Creativity***:** Users exhibited a keen sense of exploration, eagerly experimenting with PolyCreate's features to discover novel combinations and effects.

*Challenges and Problem-Solving***:** While navigating certain functionalities like material modification and animation posed challenges, users demonstrated remarkable resilience and adept problem-solving skills in overcoming obstacles.

*Tool Appreciation***:** Despite encountering hurdles, users expressed genuine appreciation for PolyCreate's versatility and adaptability, recognizing its potential as a platform for creative expression and visualization.

Feedback and Suggestions: Users provided valuable feedback on usability, offering insightful suggestions for improvement such as clearer interface cues and enhanced guidance on advanced features.

User Reflections

Post-task reflections provided users with an opportunity to articulate their experiences and insights. Many expressed satisfaction with their creative endeavors, highlighting the enjoyable and immersive nature of the task. Some acknowledged the learning curve associated with certain features but welcomed the opportunity for experimentation and skill development.

Our survey showed the majority of users rated the ease of use and understandability of LiveBuild language with a top score of 5 out of 5, indicating a high level of satisfaction and ease of comprehension.

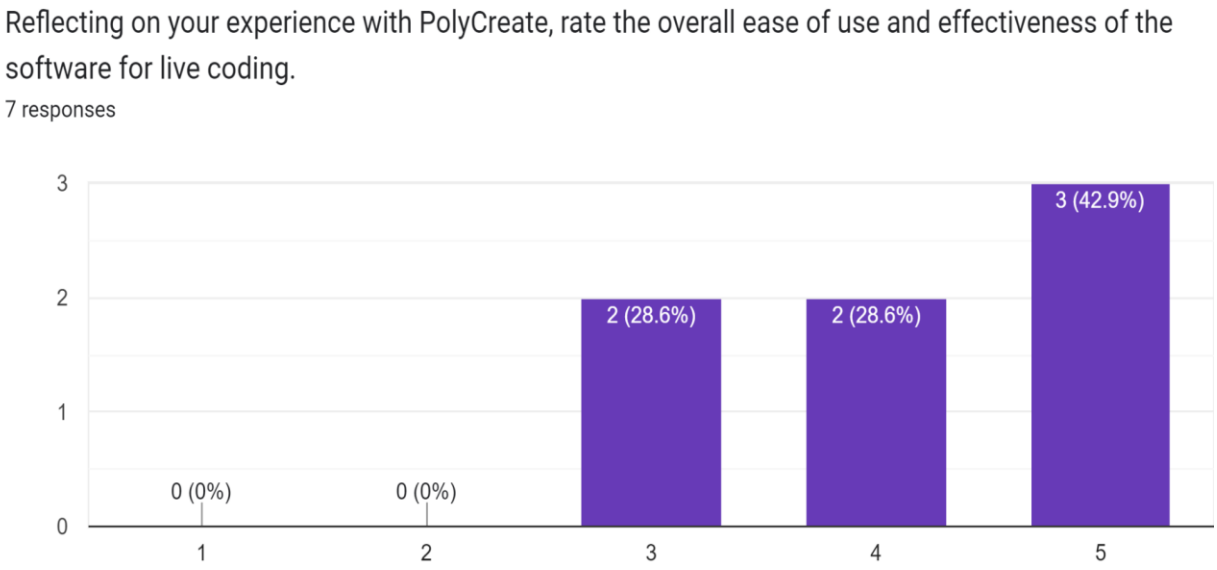

Image 17.Graph showing effectiveness of PolyCreate

When asked to reflect on the overall experience with PolyCreate, users provided slightly varied ratings, with the majority still indicating positive sentiment.

In terms of the process of modifying code and observing immediate results, the feedback was generally positive, with most users rating it 4 or 5 out of 5. This suggests that users found the environment intuitive and responsive, allowing for efficient coding and visualization

Overall, the feedback from our users provides valuable insights that will guide our efforts to refine and optimize PolyCreate to better meet the needs and expectations of

our users

.

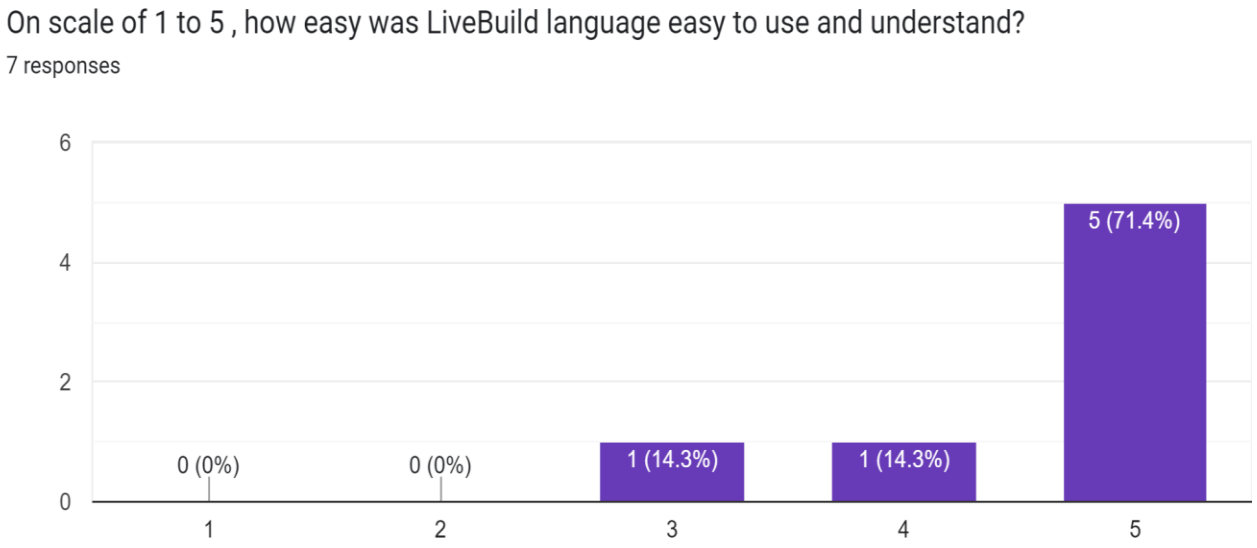

Image 18.Graph showing easiness of LiveBuild

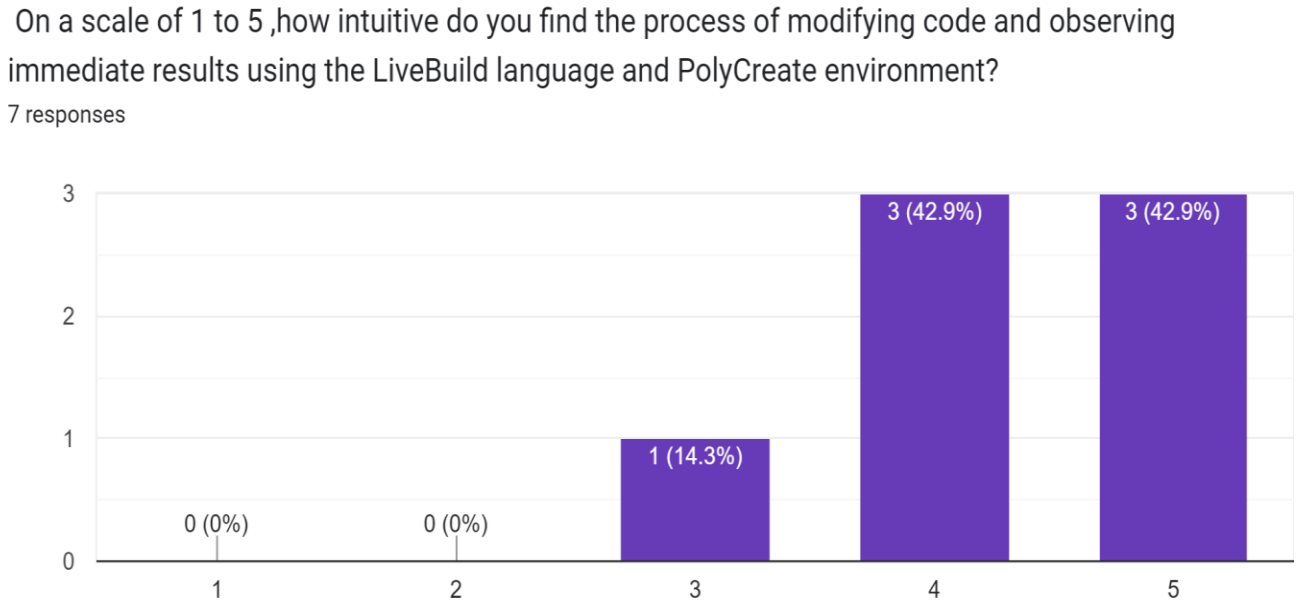

Image 19.Graph showing Intutitiveness of LiveBuild

This is what some of the users were able to make using our system and their creativity as a part of the task given to them.

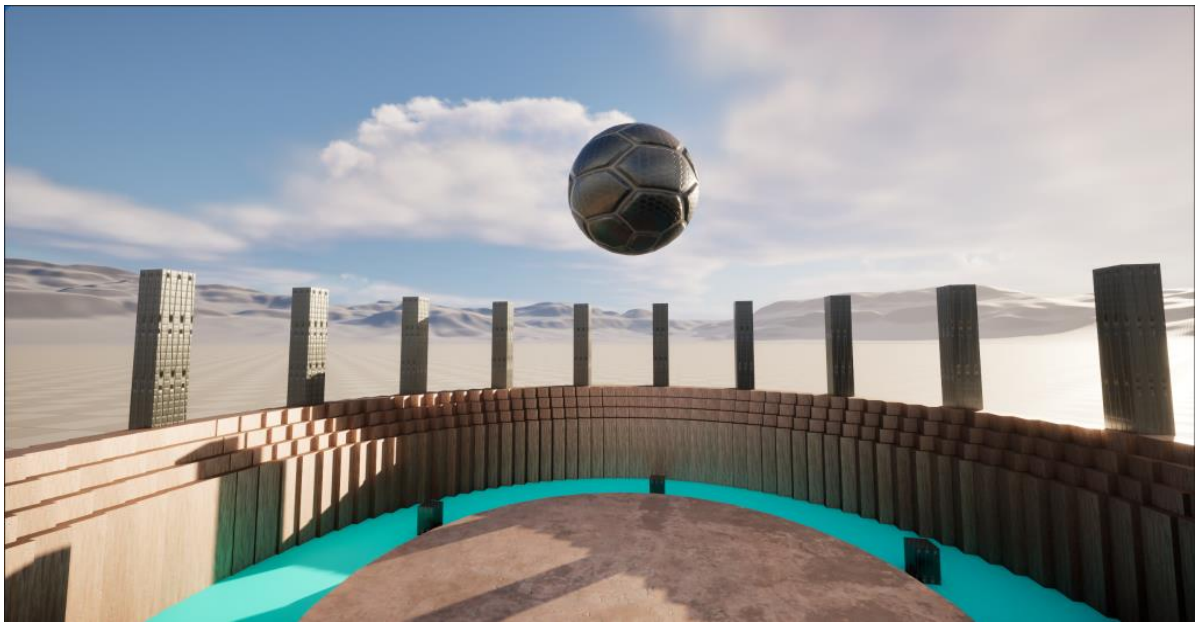

Image 20. The Arena of Death

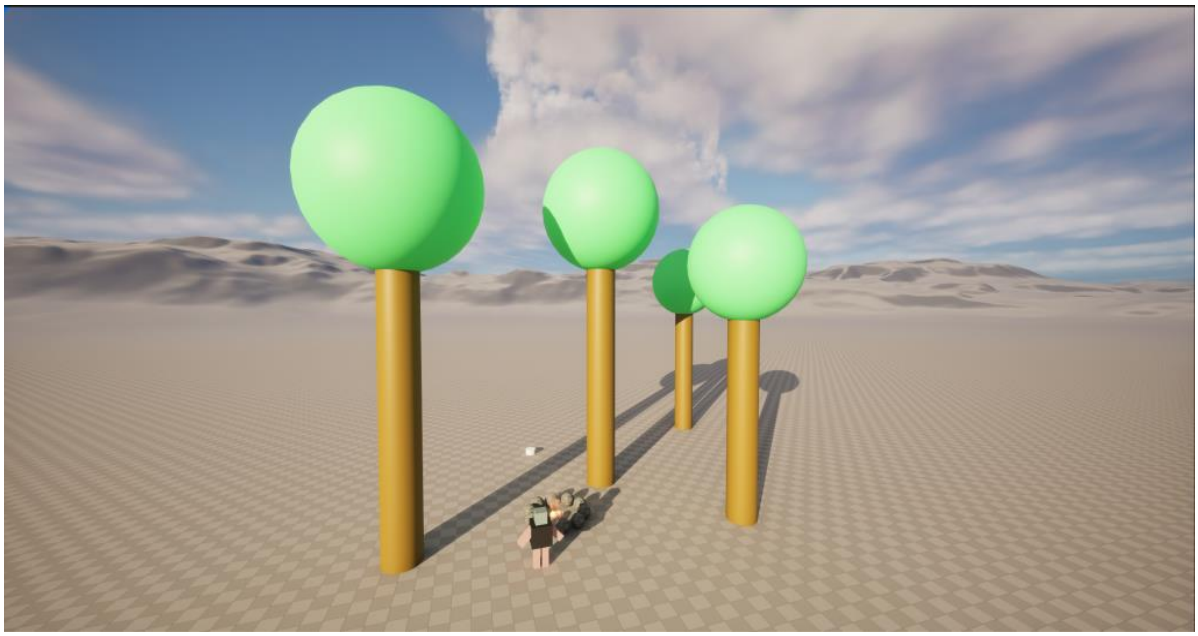

Image 21. The Forest of Solitude

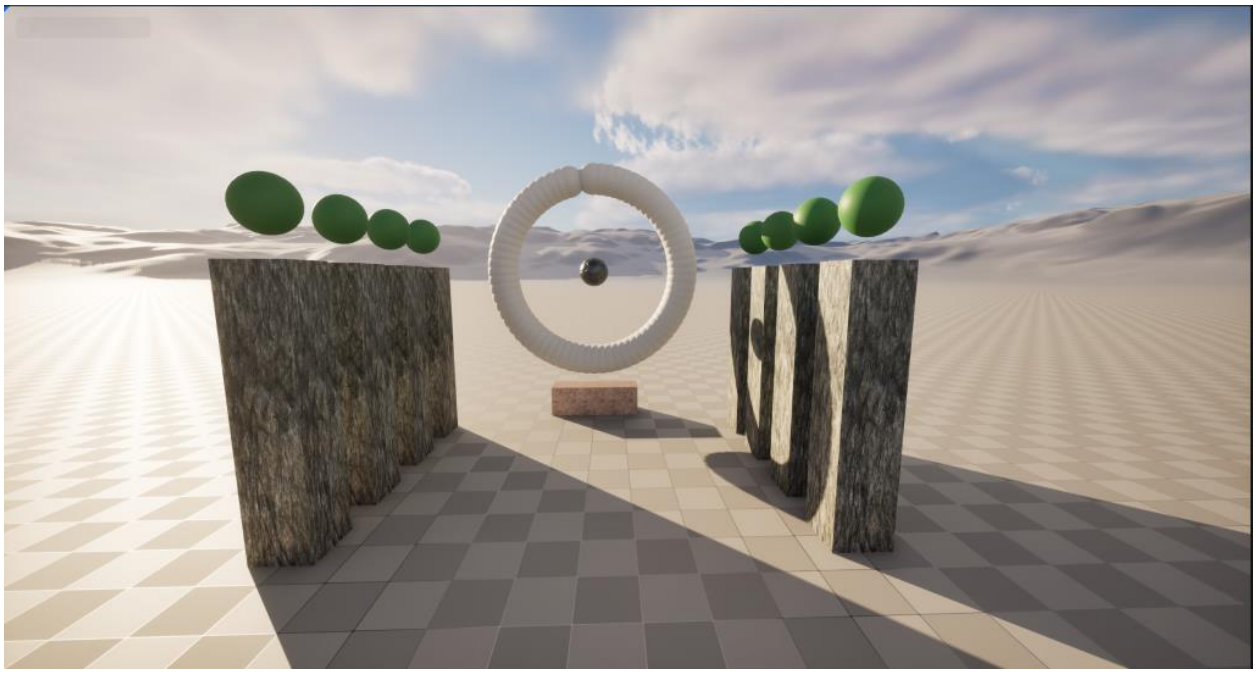

Image 22. The Altar of Creation

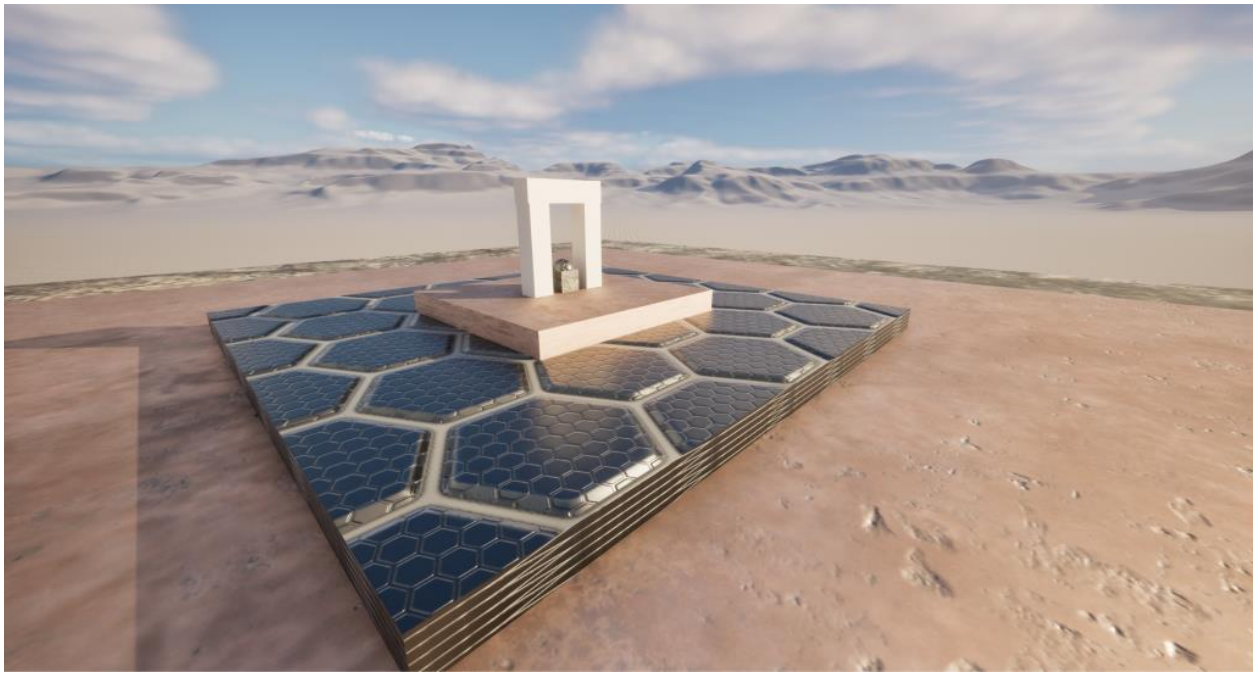

Image 22. In a galaxy, far away

5. Conclusion

The Talk Aloud Protocol sessions provided many insights into users' experiences with PolyCreate for scene-building. On average, participants spent more time using the system than initially allocated to fully explore their creative visions, indicating a deep engagement with the platform. Despite encountering a few bugs along the way, users expressed a high level of satisfaction with the overall experience. They found the LiveBuild language to be intuitive, and this included those without prior experience in live coding. One of the key things we observed was that while users spent around the first 15-20 min just getting familiar with the system. They initially found it annoying to change around the scale factors since each of the object had their size defined. But later, once they got used to it, they really got into it and tried to creatively make whatever they had in mind. on an average each user spend around 60-90 min to make what they had in mind which was almost twice the amount of allocated time for user evaluation. Users also asked for a “loop” control structure, so that they could easily carry out repetitive commands without having to enter them line by line.

We are excited to continue the development of PolyCreate. In addition to responding to requests from users, we also hope to add audio responsiveness to the system to make it more suitable for traditional live coding performances. Finally, by improving system documentation and making it easier to install, we hope to encourage potential users who are interested in the intersection of live coding performance and game engines.

6. References

[1] Bohrer, R. (2023, October). Centering Humans in the Programming Languages Classroom: Building a Text for the Next Generation. In *Proceedings of the 2023 ACM SIGPLAN International Symposium on SPLASH-E* (pp. 26-37).

[2] Wakefield, G., Roberts, C., Wright, M., Wood, T., & Yerkes, K. (2014). Collaborative live-coding virtual worlds with an immersive instrument. In *Proceedings of the 14th International Conference on New Interfaces for Musical Expression*.

[3] Kit Zellerbach and Charlie Roberts. Barbara - Live code a graphics language. https://www.barbara.graphics/

[4] Aiden Wilson. Video. 2 - way communication with OSC and Unreal(5.1) tutorial. https://www.youtube.com/watch?v=zsu1jbumTlk

[5] Geert Verhoeff . Video. Receive OSC messages with blueprints Unreal engine 5 tutorial. https://www.youtube.com/watch?v=DI2F-1iwOHA

[6] Parser generator for JavaScript. https://pegjs.org/documentation#installation-node-js'

[7] Arcadia - The integration of the Clojure Programming Language with the Unity 3D game engine. arcadia: https://github.com/arcadia-unity/Arcadia

[8] Enu – Build 3D worlds using nim in Godot game engine. https://github.com/dsrw/enu

[9] Charlie Roberts, Arthur Ames. wgsl_live. https://charlieroberts.github.io/wgsl_live/

[10] Hydra- Live coding network visuals in browser. hydra: https://hydra.ojack.xyz/

[11] C. Rodkin. (2023, July). Citation style and References Format.

[12] Lewis, Clayton. Using the "thinking-aloud" method in cognitive interface design. Yorktown Heights, NY: IBM TJ Watson Research Center, 1983

[13] Blackwell, Alan F., et al. *Live coding: a user's manual*. MIT Press, 2022

[14] Ford, B. Parsing expression grammars: a recognition-based syntactic foundation. In *Proceedings of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 111-122). 2004.

[15] Wright, M., & Freed, A. Open Sound Control: A new protocol for communicating with sound synthesizers. In *Proceedings of the International Computer Music Conference (ICMC)*. 1997.